Problem Statement
In-memory distributed databases everywhere..

What do we expect from in-memory today?

- **OLTP**
  - Improve transactional throughput
  - Ingest more per second

- **OLAP**
  - Load data faster
  - Generate reports faster
  - *Enable real-time data analytics*
Real-time Data Analytics Today

- Answer analytical queries in “real-time”
  - within a second or few second, not minutes
- Data mostly in the ~100s TB range
- Demands query execution to finish within a second or few seconds
- Queries could be ad-hoc (analytical dashboards)
- Queries could be complex
  - A few joins (star or snowflake schema)
  - Groupby, Aggregates
  - Sub-queries
Example Query 1: Big Financial Services

- Distributed Join Order, Outer Join to Inner Join rewrite

```sql
SELECT ....
FROM
    REFERRAL T1 LEFT OUTER JOIN REFERRAL_ASSIGNMENT T2
    ON T1.REFERRAL_ID = T2.REFERRAL_ID LEFT OUTER JOIN ENTITY_TO_EXTERNAL T3
    ON T1.REFERRAL_ID = T3.ENTITY_KEY
    AND T3.RELATIONSHIP_TP_CD = 1000008
    AND T3.RELATIONSHIP_OWNER = 'R' LEFT OUTER JOIN APPOINTMENT T6
    ON T6.APPOINTMENT_ID = CAST(T3.EXTERNAL_REFERENCE_KEY AS DECIMAL(10, 0))
WHERE T2.ASSIGNED_TO_CO_CC_NO = '00342|3425352'
    AND T6.START_TIME >= '2014-07-02-07.00.00.000000'
    AND T6.END_TIME <= '2014-07-03-06.59.59.000000'
```
Example Query 2: TPC-DS Q25 (Simplified)

Distributed Join Order, Bushy Joins

SELECT ....
FROM store_sales ss, store_returns sr, catalog_sales cs,
    date_dim d1, date_dim d2, date_dim d3,
    store s, item i
WHERE d1.d_moy = 4 AND d1.d_year = 2000
    AND d1.d_date_sk = ss_sold_date_sk AND i_item_sk = ss_item_sk
    AND s_store_sk = ss_store_sk AND ss_customer_sk = sr_customer_sk
    AND ss_item_sk = sr_item_sk AND ss_ticket_number = sr_ticket_number
    AND sr_returned_date_sk = d2.d_date_sk
    AND d2.d_moy BETWEEN 4 AND 10 AND d2.d_year = 2000
    AND sr_customer_sk = cs_bill_customer_sk AND sr_item_sk = cs_item_sk
    AND cs_sold_date_sk = d3.d_date_sk
Query Optimization Frequency

- Ad-hoc queries from analytical dashboards always require optimization.

- Queries that are not ad-hoc may also require optimization:
  - The first invocation of the query
  - If the data statistics have changed significantly
  - If the query parameters differ from previous invocations
Why is optimization time important?

- Query Optimization time cannot afford to be the bottle-neck in real-time analytics
- Very small time budgets (<100ms) for query optimization
- Optimizer still should be able to produce efficient execution plans with near-optimal runtime performance

Query Optimization has the potential to become the bottleneck in real-time analytics
MemSQL Query Optimizer
Overview of MemSQL

- Fully distributed in-memory database system
- Supports both OLTP and OLAP workloads
- Extreme performance on commodity hardware
- Designed for infinite scale-out
- Two-tier architecture; scalability on every tier
MemSQL Query Optimizer

- A modular and flexible query optimizer
- Built from scratch using a lot of C++ lambda functions
- Three principal components
  - Rewriter
  - Enumerator
  - Planner
MemSQL Query Optimizer

- **Rewriter:**
  - Applies query rewrites based on heuristics or cost;
  - Costs rewrites by calling the Enumerator
  - Interleaves mutually beneficial rewrites

- **Enumerator**
  - Join order based on distributed cost
  - Data movement decisions (e.g. broadcast, reshuffle)

- **Planner**
  - Converts the chosen logical execution plan to a sequence of distributed query and data movement operations.
Where does the optimizer spend time?

- Time consuming components need to be dealt with efficiently and intelligently
- Cost-based query rewrites
- Join enumeration to choose best join order
  - Generating bushy join plans
- Distributed join order
  - Search space analysis
Reducing Query Optimization Time
Generation of Bushy Plans outside Enumerator

- Considering all bushy joins in join enumeration is extremely expensive
- However, bushy joins are critical for execution performance
  - E.g. several TPC-DS queries benefit by 3-10x
- Consider only promising bushy joins instead of all possible cases
- Look for common query shapes that benefit from bushy plans and introduce bushiness via query rewrite
Extremely Fast Enumeration

- Prune heavily to eliminate a huge majority of the search space
- Enumerator uses several heuristics to generate initial candidate join orders;
  - Cost each candidate join order
  - Cheapest candidate provides an initial upper bound on the cost
- Details are in the paper
Some Experimental Results
## Optimization time for TPC-H Queries

<table>
<thead>
<tr>
<th>Query</th>
<th>Tables</th>
<th>Time (ms.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q3</td>
<td>3</td>
<td>5.09</td>
</tr>
<tr>
<td>Q5</td>
<td>6</td>
<td>9.99</td>
</tr>
<tr>
<td>Q7</td>
<td>6</td>
<td>5.94</td>
</tr>
<tr>
<td>Q8</td>
<td>8</td>
<td>20.7</td>
</tr>
<tr>
<td>Q9</td>
<td>6</td>
<td>6.36</td>
</tr>
<tr>
<td>Q21</td>
<td>6</td>
<td>11.02</td>
</tr>
</tbody>
</table>

**Minimal optimization time for most queries**
Pruning Percentages for TPC-H Queries

<table>
<thead>
<tr>
<th>Query</th>
<th>Tables</th>
<th>Pruning %</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q3</td>
<td>3</td>
<td>25.00%</td>
</tr>
<tr>
<td>Q5</td>
<td>6</td>
<td>61.46%</td>
</tr>
<tr>
<td>Q7</td>
<td>6</td>
<td>72.92%</td>
</tr>
<tr>
<td>Q8</td>
<td>8</td>
<td>95.80%</td>
</tr>
<tr>
<td>Q9</td>
<td>6</td>
<td>84.90%</td>
</tr>
<tr>
<td>Q21</td>
<td>6</td>
<td>62.50%</td>
</tr>
</tbody>
</table>

Pruning percentage huge for most queries
## Bushy Join Speedup and Overhead for TPC-DS

<table>
<thead>
<tr>
<th>Query</th>
<th>Optimization Overhead</th>
<th>Execution Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q15</td>
<td>13%</td>
<td>5.8x</td>
</tr>
<tr>
<td>Q25</td>
<td>16%</td>
<td>10.1x</td>
</tr>
<tr>
<td>Q46</td>
<td>12%</td>
<td>2.85x</td>
</tr>
</tbody>
</table>

**Significant execution speedup with minimal optimization overhead**
Current Work
Work in progress..

- Parallelizing the join enumeration process
- Refine heuristics based on knowledge from customer experiences
- Getting the costing “right”
Q & A
Thank You

www.memsql.com